Node.js

华清远见-成都中心-H5教学部

第38章 客户端与服务器

华清远见-成都中心-H5教学部

目录

1、终端、客户端与服务器

1. **终端：** 也叫终端设备，是指实物，如笔记本、手机、平板、电视机、自助售货机、无接触取货机、打印机、指纹打卡机、人脸打卡机等。

2. **客户端：** 是指为客户提供本地服务的应用程序。比如：浏览器，电子邮件客户端，但是客户端功能的实现，往往需要与服务器相互配合，接受服务器的服务。

3. **服务器：** 是为客户端提供服务的应用程序。根据业务的服务类型不同，可分为文件服务器，数据库服务器，应用程序服务器，web服务器，视频服务器等。

1、终端、客户端与服务器

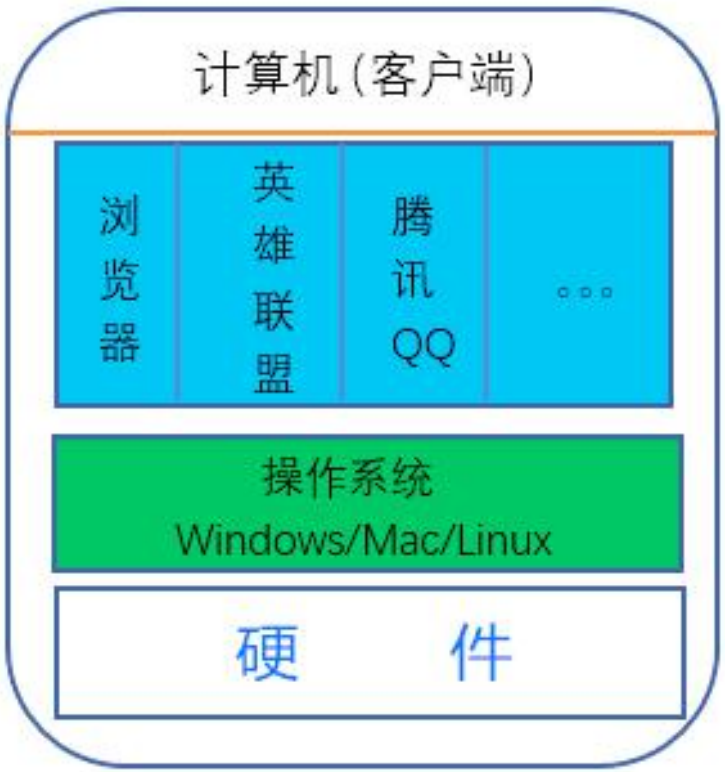
终端及安装的客户端

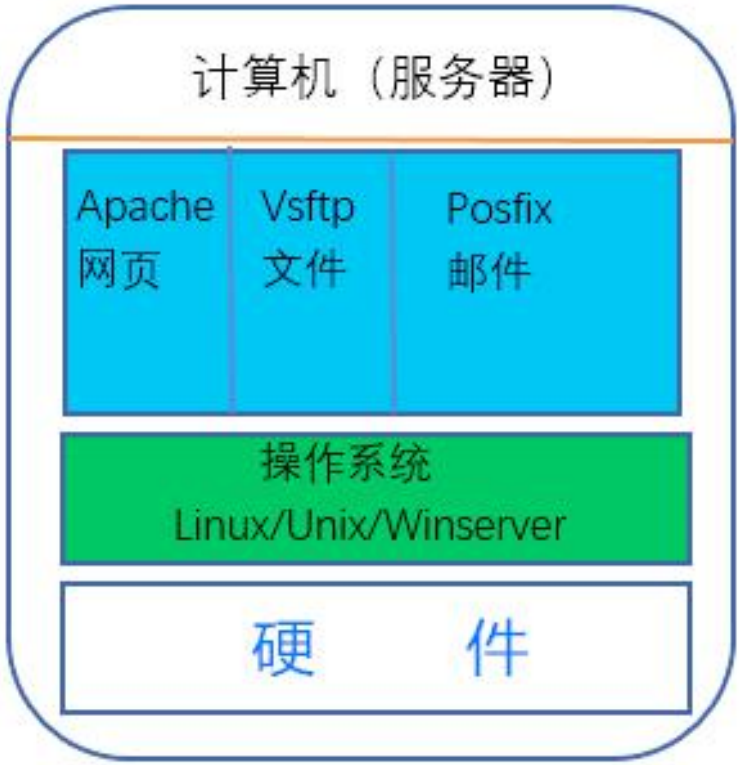
1、终端、客户端与服务器

服务器

1、终端、客户端与服务器

一般情况下客户端软件和服务器软件安装在不同的计算机设备上。





1、终端、客户端与服务器

1. 客户端电脑上安装的是我们日常使用的应用软件。

   如QQ，音乐播放器，视频播放器，浏览器等。这些应用都可以被称为客户端。

2. 服务器电脑上安装的是为客户端提供服务的软件。

   如提供网页服务的应用有：Apache/Nginx/Tomcat/IIS等；

   提供文件下载和上传服务的有vsftp等；

   提供邮件服务的应用有posfix等。

1、终端、客户端与服务器

B/S：使用H5技术开发网页应用部署到服务器上，用户在浏览器上访问，这种体系结构就是B/S (Browser/Server)结构。

C/S：使用H5技术开发APP、小程序等应用，服务器端自然是放到服务器上，这种体系结构就是C/S(Client/Server)结构。

2、计算机通信基础

通信协议： 是指交流的双方约定好的规则。

终端设备通过各种协议(**Protocol**)进行通信，其中互联网协议是互联网的核心。

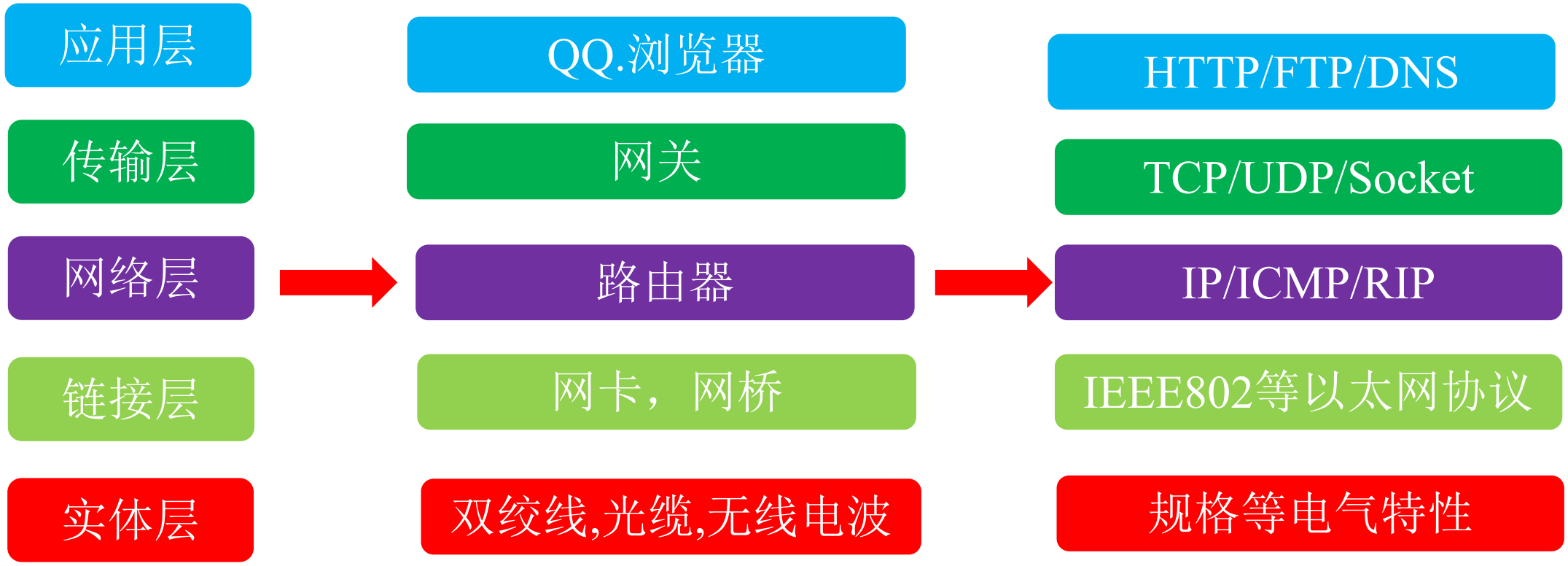
人与人：
日语
普通话
英语
韩语

计算机与计算机
HTTP
TCP/IP
UDP
DNS

程序员与计算机：
C语言
Java
JavaScript
C#
C++
PHP

2、计算机通信基础

采用层次性的结构模型，将网络分成若干层次，每个层次负责不同的功能。有的分成七层，有的分五层。每一层都有要遵循一定的协议。



2、计算机通信基础-实体层

实体层：

用光缆，无线电波，双绞线等物理手段将通信的计算机连接起来。其中传输的是0和1两种电信号。

2、计算机通信基础-链接层

链接层：

使用以太网协议将0和1电信号进行分组，一定数量的电信号组成一个数据包。这种数据包包含标头和数据两部分，格式如下：

以太网标头

Data是发送的数据，Head中包含了数据的发送者和接收者。

2、计算机通信基础-链接层

链接层：

以太网协议规定，连入网络的所有设备，都必须具有“网卡”接口。数据包必须是从一块网卡传送到另一块网卡。网卡的地址，就是数据包的发送地址和接收地址，叫做MAC地址。每块网卡都有一个全球唯一的MAC地址。

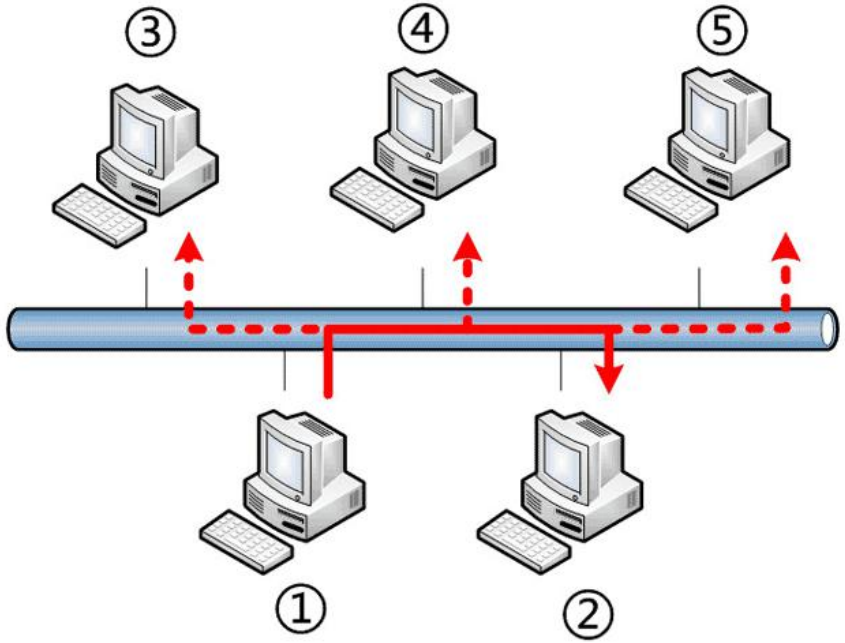
```
以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Intel(R) Ethernet Connec
   物理地址. . . . . . . . . . . . . : C8-5B-76-8B-CB-F5
```

2、计算机通信基础-链接层

链接层：

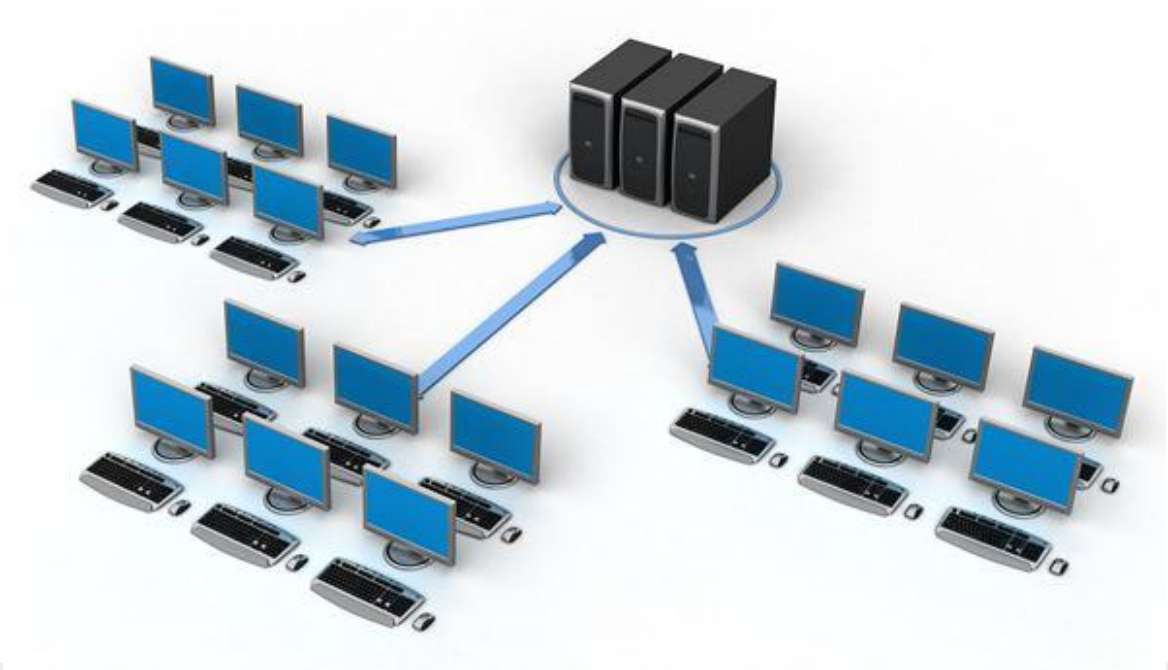
以太网通过“广播”的方式发送数据，在本网络中其他所有计算机都可以接收到其他计算机发出来的数据包。这些计算机读取数据包中的head中的接收方的Mac地址，和自己的Mac地址比较，如果相同，说明是发送给自己的。如果不同就将数据包丢弃。



2、计算机通信基础-网络层

网络层：

只有链接层的话，只能在局域网的范围内发送数据。互联网是由无数个局域网组成的巨型网络。如果要给局域网之外的计算机发送数据，还需要网络层。



2、计算机通信基础-网络层

网络层：

网络层引进了一套新的地址，使得我们能够区分不同的计算机是否属于同一个局域网络。这套地址就叫做"网络地址"，简称"网址"。

网址帮助我们确定计算机所在的局域网络，MAC地址则将数据包送到该局域网中的目标网卡。

规定网络地址的协议，叫做IP协议。它所定义的地址，称为IP地址。广泛采用的是IP协议第四版，简称IPv4。该协议规定，网络地址由32个二进制位组成。

10101100.00010000.11111110.00000001

172.16.254.1

习惯上用分成四段的十进制数表示IP地址，从0.0.0.0一直到255.255.255.255

2、计算机通信基础-网络层

网络层：

由于IP地址是数字标识，使用时难以记忆和书写，因此在IP地址的基础上又发展出一种符号化的地址方案，来代替数字型的IP地址。每一个符号化的地址都与特定的IP地址对应，这样网络上的资源访问起来就容易得多了。这个与网络上的数字型IP地址相对应的字符型地址，就被称为域名。

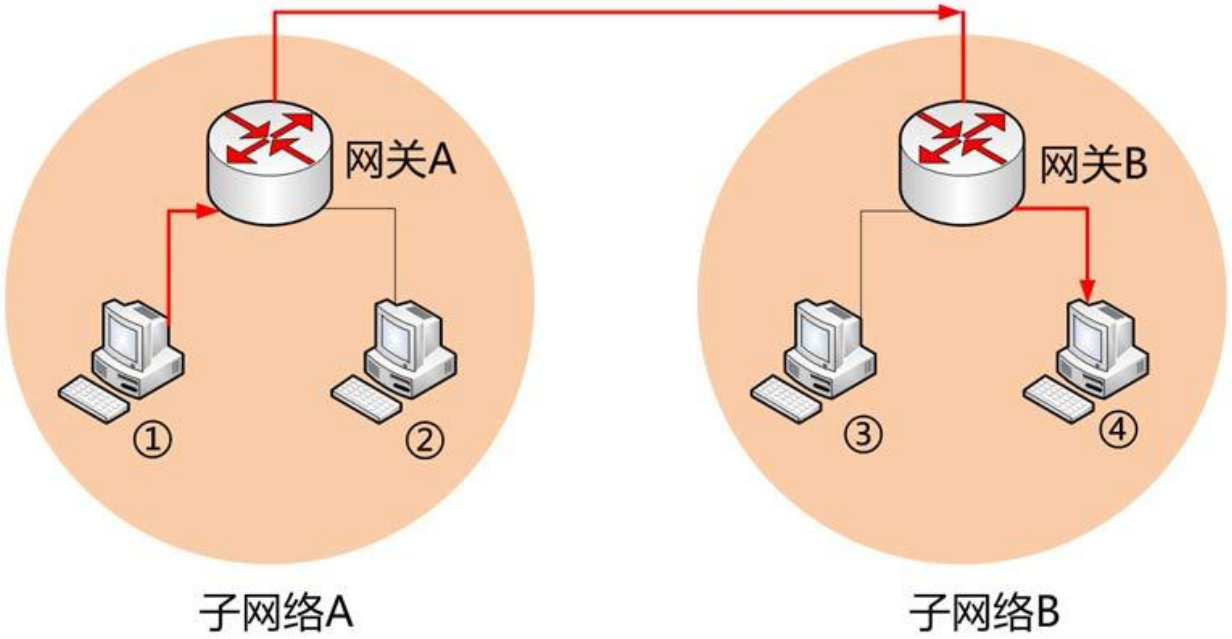
域名虽然便于人们记忆，但机器之间只能互相认识IP地址，它们之间的转换工作称为域名解析，域名解析需要由DNS协议来完成。

注意：域名和IP不一定一一对应，可能多个域名都指向同一个IP。

2、计算机通信基础-网络层

网络层：

向局域网外的计算机发送数据包的时候，事实上我们是没有办法获取对方的Mac地址的，只能把数据包传送到两个局域网络连接处的"网关"（gateway），让网关去处理。



2、计算机通信基础-网络层

网络层：

以太网的数据包经过网络层，会添加上包含IP地址，长度等信息的标头。变成了如下的样子：

有了MAC地址和IP地址，我们已经可以在互联网上任意两台主机上建立通信。

2、计算机通信基础-传输层

传输层：

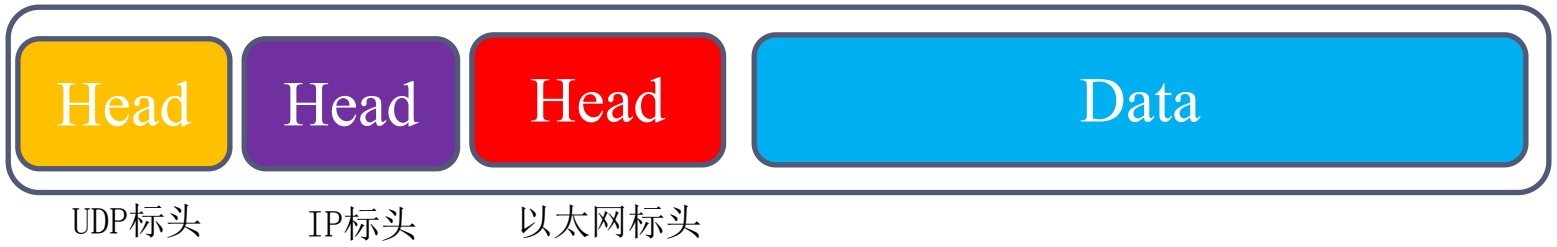
当一个数据包从互联网上发过来的时候，我们怎么区分它是表示网页的内容，还是表示文件下载的内容呢？是“端口”。端口就是每一个使用网卡的程序的编号。数据包都发到主机的特定端口，所以不同的程序就能取到自己所需要的数据。

“传输层”的功能，就是给数据包添加端口信息。使用的协议有TCP和UDP。端口号的范围从0到65535。0-1023都是系统程序占用的，我们不能使用。

2、计算机通信基础-传输层

传输层：

以太网的数据包经过传输层，会添加上包含端口信息的标头。变成了如下的样子：



数据包经过链接层，网络层，传输层之后，就可以成功发送到其它计算机中的特定应用程序中。

2、计算机通信基础-应用层

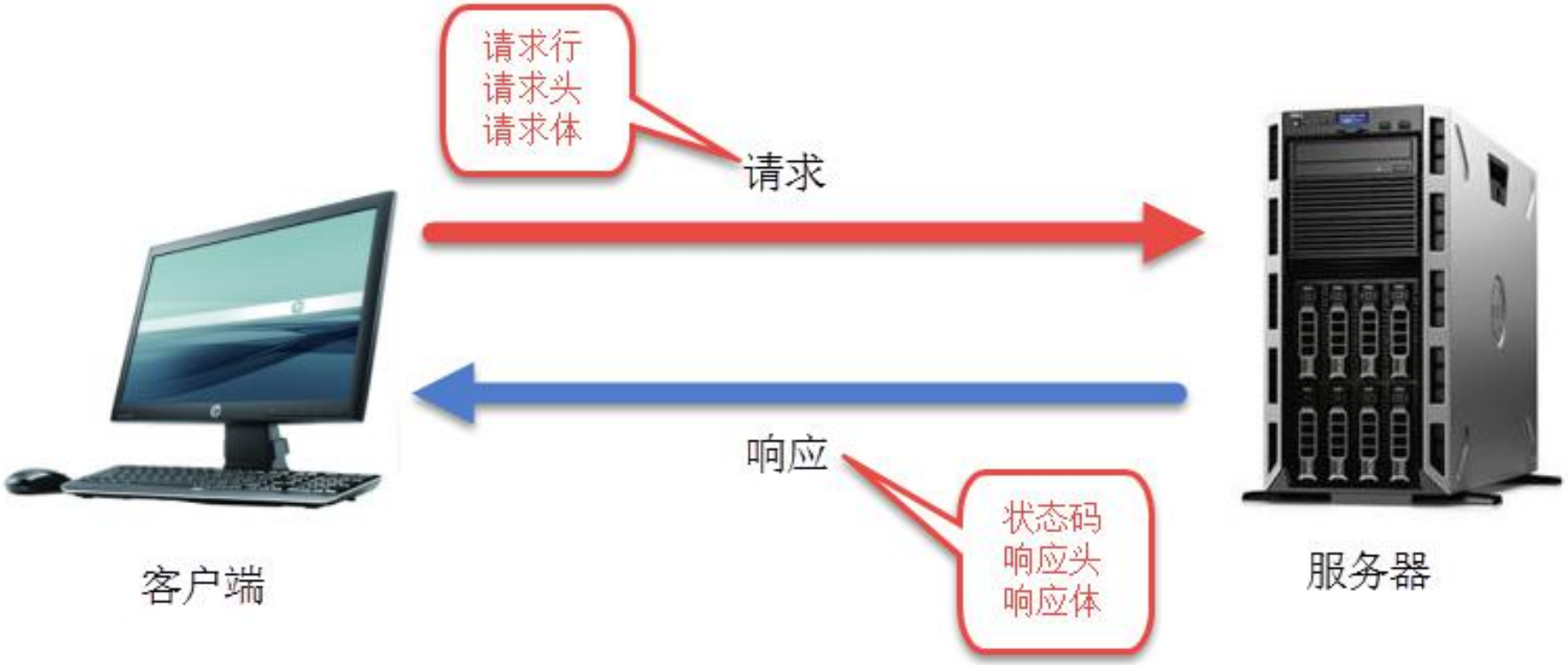
应用层：

互联网上传递的数据包五花八门，有网页，文件，邮件等等。需要按照特定的协议解析这些数据包，我们才能知道它们是什么类型的数据。比如解析文件数据的协议是FTP，解析网页数据的协议是HTTP，这些协议构成了应用层。

3、HTTP协议

HTTP协议（HyperText Transfer Protocol 超文本传输协议）用于服务器端和客户端的通信。是B/S模式开发接触得最多的一种协议。

HTTP协议永远都是客户端发起请求，服务器响应请求。



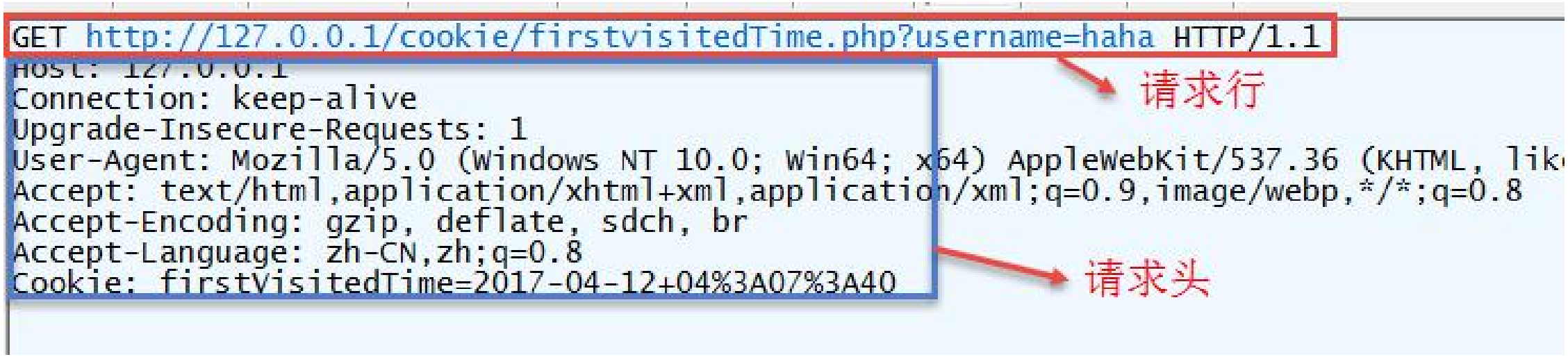
3、HTTP协议

使用HTTP进行的网络连接是一种没有状态的连接，当服务器响应了之后，就会于客户端断开连接。并且不会记录客户端的任何信息。

如果浏览器访问的网页中使用了单独的CSS，JavaScript文件。那么会发出多次HTTP请求。涉及到多少个文件，就有多少个请求。

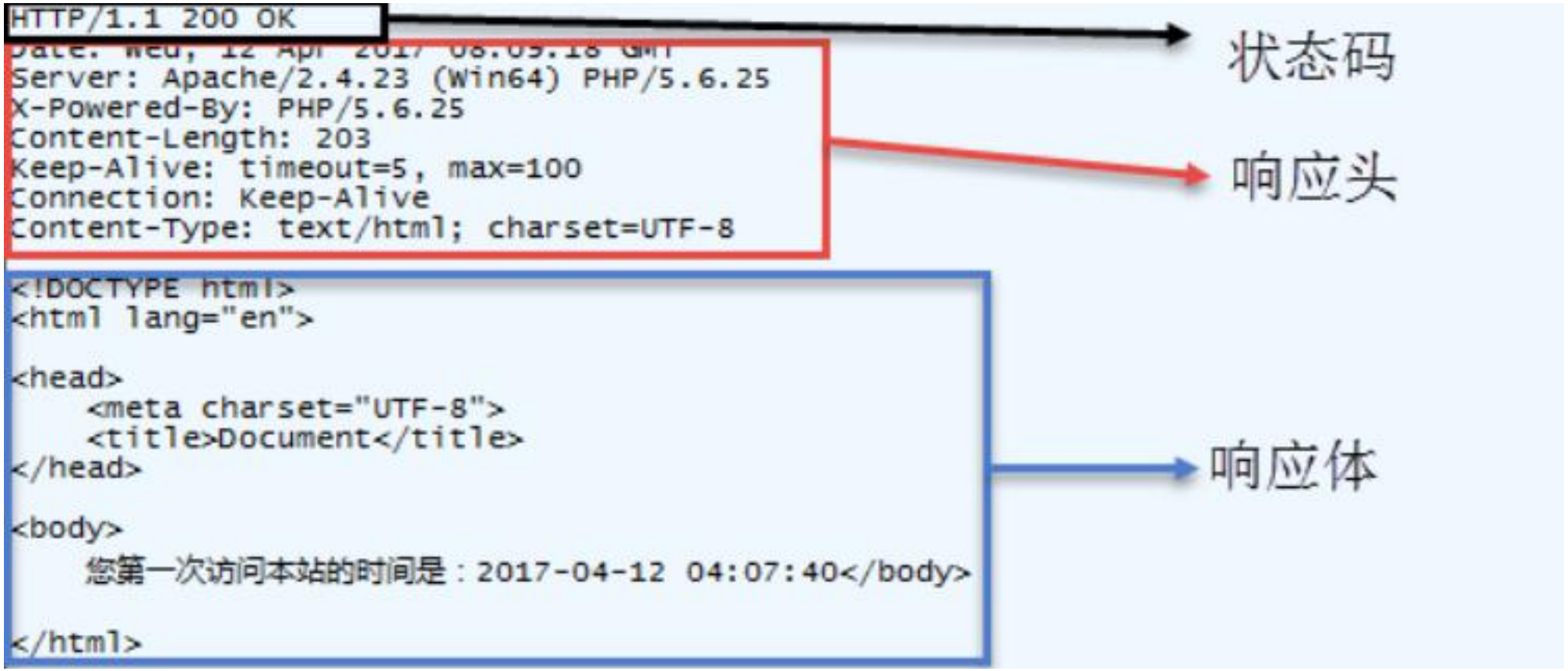
3、HTTP协议

HTTP协议请求格式：



3、HTTP协议

HTTP协议响应格式：



3、HTTP协议

Connection之keep-alive：

第一步，Client发出request，使用的是HTTP协议的1.1版本，默认keep-alive，开启长连接；

第二步，服务器收到request中的HTTP协议为1.1，把它当做一个长连接请求，其将在response的header中增加“Connection： keep-alive”。此时就不会关闭已建立的tcp连接；

第三步，Client收到Server的response中包含“Connection:keep-alive”，就当做一个长连接，不会关闭tcp连接，并且在后续的request中继续使用此连接;

3、HTTP协议

Connection之keep-alive相关参数：

Keep-Alive: timeout=5, max=100

Timeout：过期时间5秒；

Max：最多一百次请求，强制断掉连接；

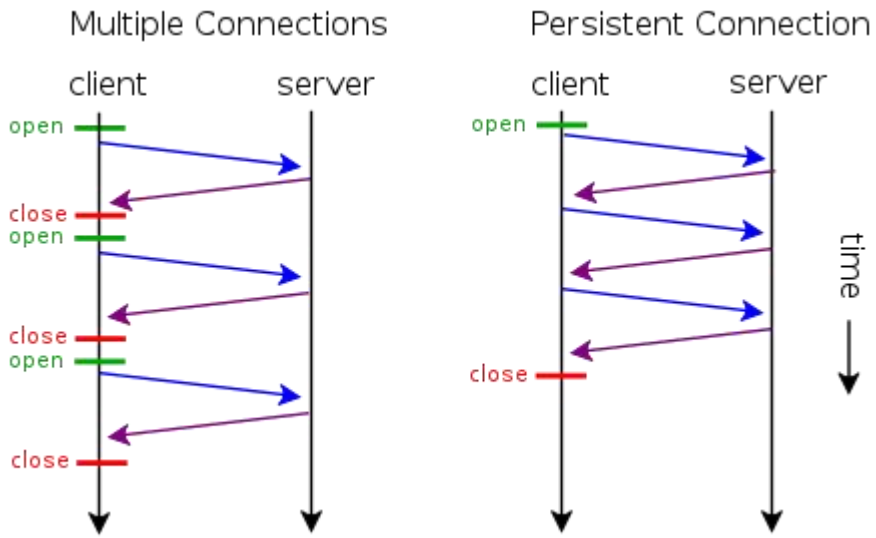
Keep-alive是否开启以及超时时间和最大请求次数在服务器端都是可以配置的：

如Apache的配置如下：

```
KeepAlive  On
MaxKeepAliveRequests  100
KeepAliveTimeout  5
```

3、HTTP协议

长连接性能更优：



3、HTTP协议

Request之Accept-Encoding：

Accept-Encoding 是浏览器发给服务器，声明浏览器支持的编码类型的；

常见的有

```
Accept-Encoding: compress, gzip                          //支持compress 和gzip类型
Accept-Encoding:                                         //默认是identity
Accept-Encoding: *                                       //支持所有类型
Accept-Encoding: compress;q=0.5, gzip;q=1.0              //按顺序支持 gzip , compress
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0      // 按顺序支持 gzip , identity
```

3、HTTP协议

Cache-Control ：

Response之 Cache-Control:max-age=3600；表示一分钟之内使用缓存数据

常见的有

no-cache：强制每次请求直接发送给源服务器，而不经过本地缓存版本的校验；

max-age>0 时，直接从游览器缓存中提取 ；

max-age<=0 时，向server 发送http请求确认 ,该资源是否有修改：有的话 返回200 ,无的话 返回304；

3、HTTP协议

Last-Modified ：

文件在服务器的最后修改时间；

Expires ：

用于控制请求文件的有效时间，当请求数据在有效期内时客户端浏览器从缓存请求数据而不是服务器端。

当缓存中数据失效或过期，才决定从服务器更新数据。

3、HTTP协议

Cache-Control: max-age=秒 和 Expires ：

Expires = 时间，HTTP 1.0 版本缓存的载止时间，允许客户端在这个时间之前不发送请求；
max-age = 秒，HTTP 1.1版本，资源在本地缓存多少秒；

如果max-age和Expires同时存在，则被Cache-Control的max-age覆盖；

Expires 返回的到期时间是服务器端的时间，如果客户端的时间与服务器的时间相差很大，那么误差就很大
所以在HTTP 1.1版开始，使用Cache-Control: max-age=秒替代；
Expires =max-age + “每次下载时的当前的request时间”
所以一旦重新下载的页面后，expires就重新计算一次，但last-modified不会变化 ；

3、HTTP协议

响应状态码：

表示服务器对请求的各种不同处理结果和状态，是一个三位的十进制数。可以分为以下5类：

状态码	含义
100~199：	接收请求成功
200~299	接收请求并处理完成。
300~399	请求需进一步细化。
400~499	客户端请求有错误
500~599	服务器端出现错误

3、HTTP协议

经常遇到的状态码：

200：一切正常。

404：服务器上不存在客户端请求的资源。

500：服务器内部错误。

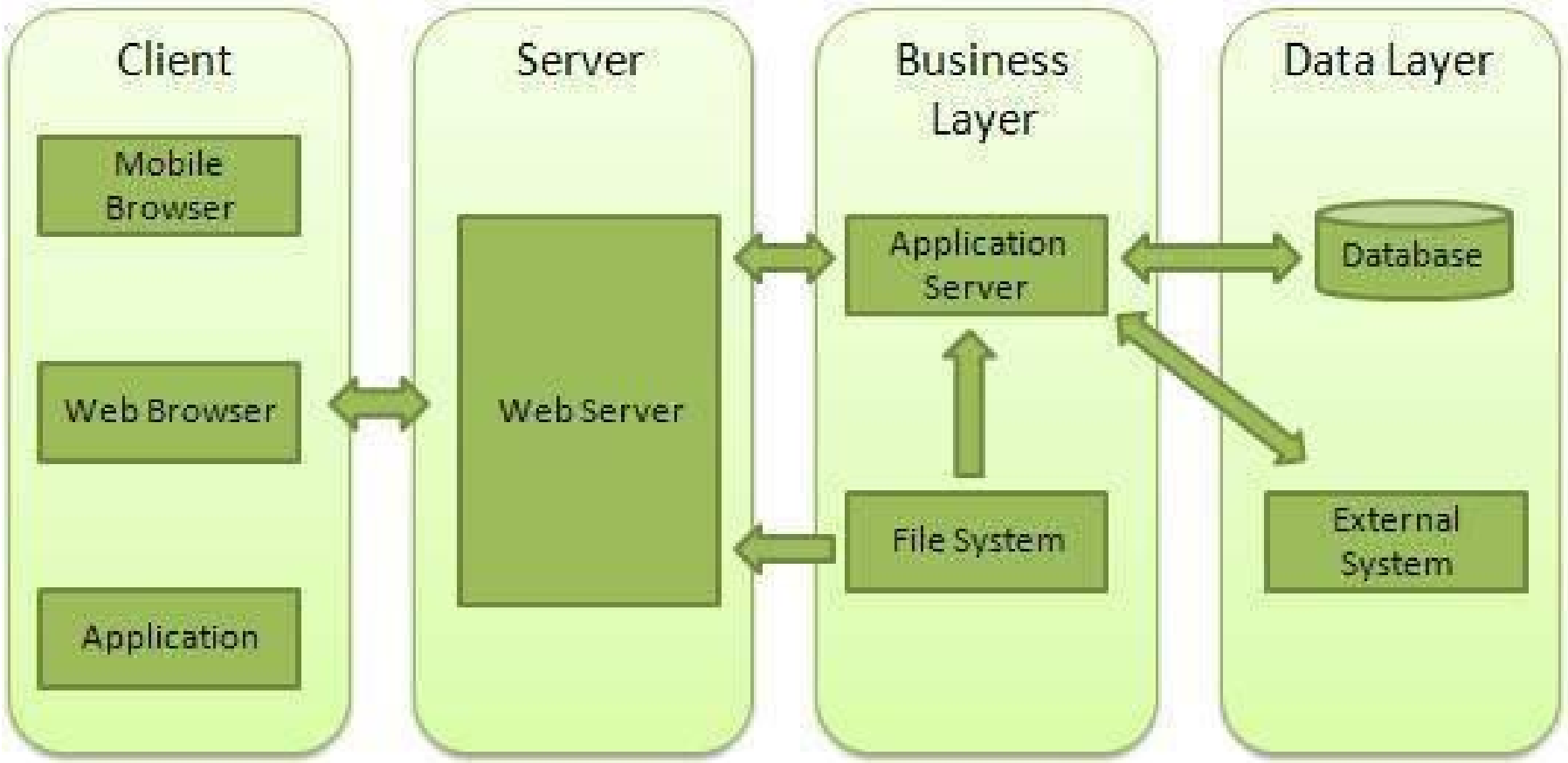
4.web服务器

web服务器一般指网站服务器，是指驻留于因特网上某种类型计算机的程序，web服务器的基本功能就是提供web信息浏览服务。它只需要支持HTTP协议、HTML文档格式及URL，与客户端的网络浏览器配合。

大多数web服务器都支持服务端脚本语言（ JAVA 、PHP、C++ ）等，并通过脚本语言从数据库获取数据，将结果返回给客户端浏览器。

目前主流的web服务器软件有：Apache、Nginx、Tomcat、IIS。

4.web服务器的应用架构



4.web服务器的应用架构

Client - 客户端，一般指浏览器，浏览器可以通过 HTTP 协议向服务器请求数据。

Server - 服务端，一般指 Web 服务器，可以接收客户端请求，并向客户端发送响应数据。

Business - 业务层， 通过 Web 服务器处理应用程序，如与数据库交互，逻辑运算，调用外部程序等。

Data - 数据层，一般由数据库组成

4.web服务器的工作原理

1. **连接过程---连接服务器**

   就是Web服务器和其浏览器之间所建立起来的一种连接。

2. **请求过程---请求服务器数据**

   请求过程就是Web的浏览器运用socket这个文件向其服务器而提出各种请求。

3. **应答过程---服务器处理请求并做响应**

   应答过程运用HTTP协议把在请求过程中所提出来的请求传输到Web的服务器，进而实施任务处理，然后运用HTTP协议把任务处理的结果传输到Web的浏览器，同时在Web的浏览器上面展示上述所请求之界面。

4. **关闭连接---断开服务器连接**

   关闭连接就是当上一个步骤--应答过程完成以后，Web服务器和其浏览器之间断开连接之过程。

Web服务器上述4个步骤环环相扣、紧密相联，逻辑性比较强，可以支持多个进程、多个线程以及多个进程与多个线程相混合的技术。

第39章 Node.js基础

华清远见-成都中心-H5教学部

目录

Node.js概述

一． Chrome的V8 引擎可以不在浏览器

二． Node.js®是基于Chrome的V8 JavaScript引擎构建的JavaScript运行环境。

三． Node.js与javaEE,PHP不同，node.js不运行在任何web服务器软件基础之上

四． Node.js最大的特点是: 单线程、非阻塞异步I/O、事件驱动

五． 应用场景: 在硬件条件差并且I/O操作多的功能下，追求最高的并发，更好的性能

Node.js开发环境

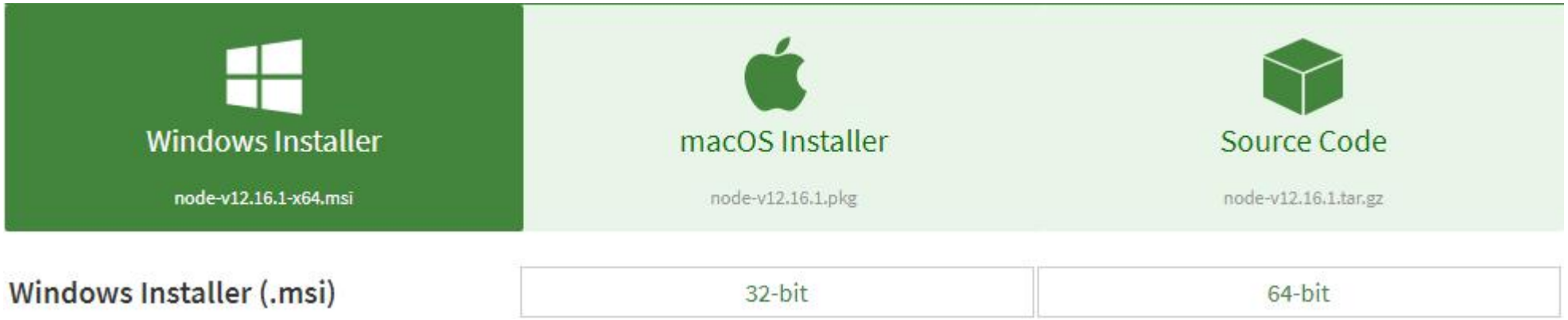
一．官网下载

Node.js官网地址：https://nodejs.org/en/download/　下载nodejs；

Node.js中文地址：http://nodejs.cn/；

Nodejs学习文档：https://nqdeng.github.io/7-days-nodejs/#1.1

根据你的操作系统去下载即可，下载msi后缀的，直接双击安装。



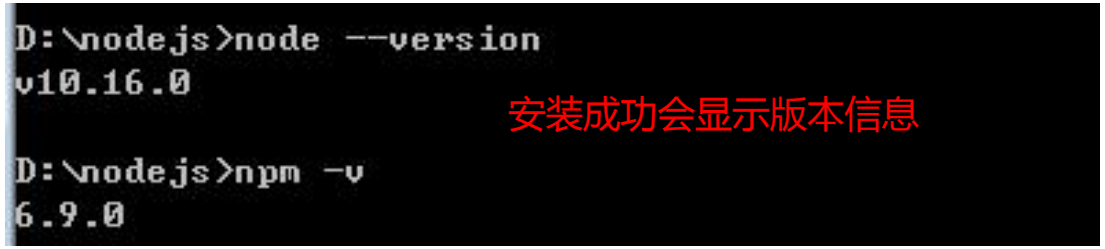
Node.js开发环境

二． 安装Node.js

1. 安装路径一般不要指定到系统盘，安装路径越简单越好，路径不要出现**空格、中文或特殊字符**等，这里指定的路径是：D:\nodejs，你完全可以指定你想要的路径，**最好不要装在系统盘，尤其是win10的系统，会有很多权限问题**；

   在接下来的配置中，一定要根据你的安装路径进行参数设置；

2. 测试是否安装成功：



Node.js开发环境

三． 配置环境变量

1. 配置npm的全局模块安装路径及cache路径：在nodejs主目录(我这里是D:\nodejs)下创建两个文件夹：node_global、node_cache，然后执行如下命令：

   npm config set prefix "D:\nodejs\node_global"

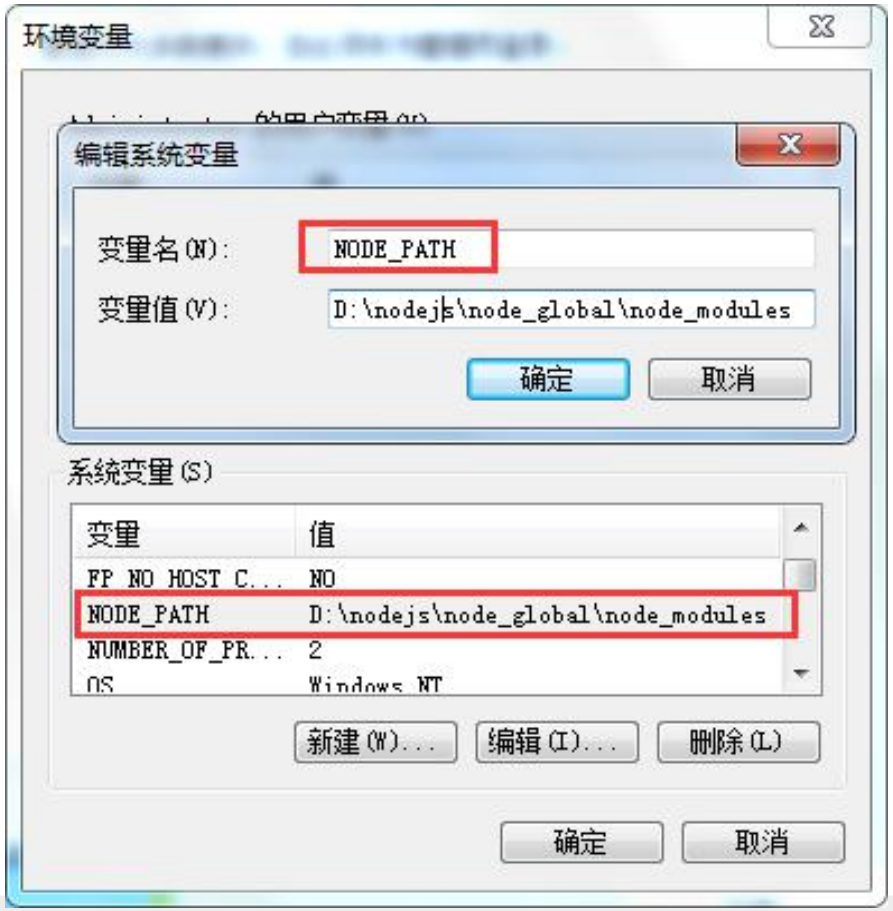
   npm config set cache "D:\nodejs\node_cache"

2. 我的电脑 → 右键→属性→高级系统设置→环境变量→系统变量

   中添加系统变量：

   变量名：NODE_PATH

   变量值：D:\nodejs\node_global\node_modules



Node.js开发环境

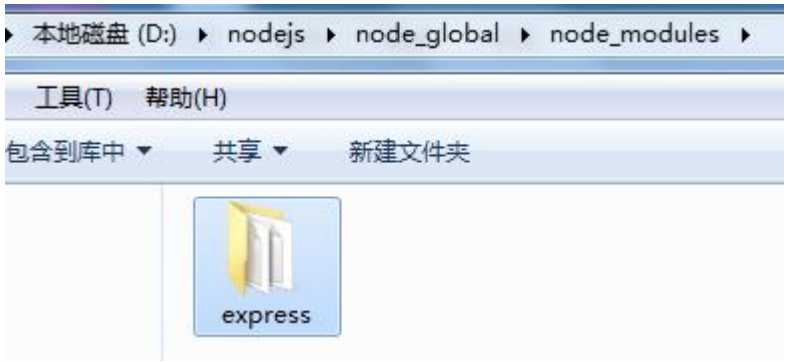
三．配置环境变量

3. 在命令行输入以下命令安装express包： 一定要在上一步配置完成后重启一次DOS命令行

```
npm install express –g
```

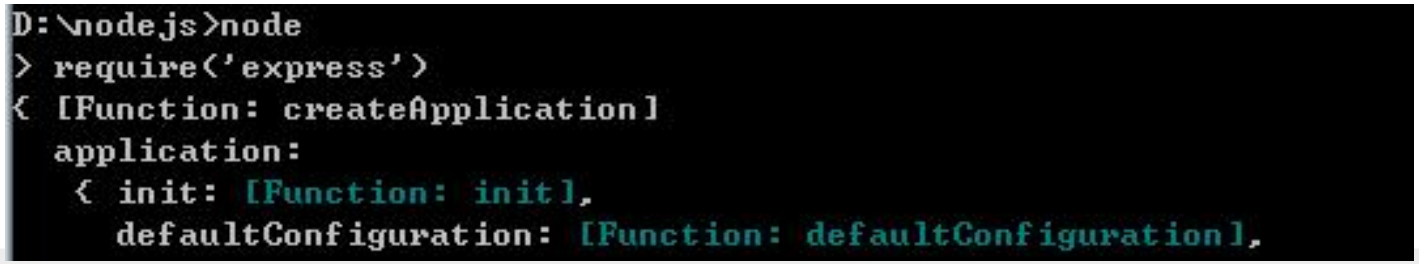
安装完成后到D:\nodejs\node_global\node_modules看是否有该包；

如图表示成功：



4. 在命令行输入node进入编辑模式，输入以下代码检查是否能正常加载模块：require(“express”);

如图表示一切OK：



Node.js开发环境

四． 配置淘宝cnpm

1. 安装淘宝cnpm：

```
npm  install –g  cnpm --registry=https://registry.npm.taobao.org
```

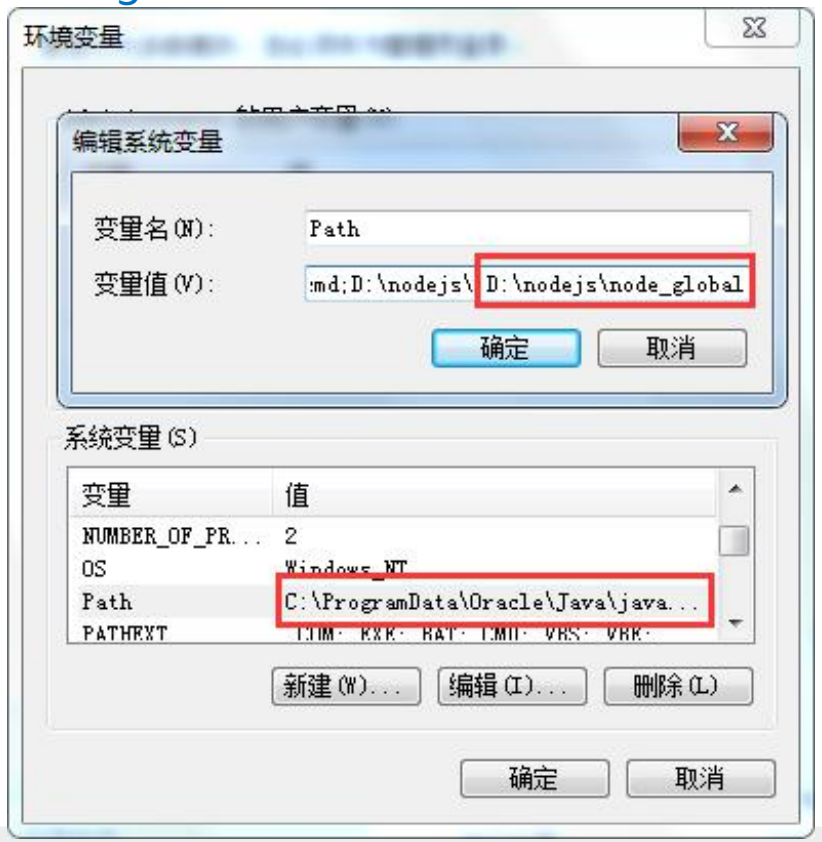
2. 添加系统变量path的内容：

因为cnpm会被安装到D:\nodejs\node_global下，而系统变量path并未包含该路径。

需要注意的是，win7下多个路径之间使用分号(;)隔开，最后一个路径后面不要分号，如：

D:\Program Files\Git\cmd;D:\nodejs\;D:\nodejs\node_global

win10和win7这里添加方法不一样



Node.js开发环境

四．配置淘宝cnpm

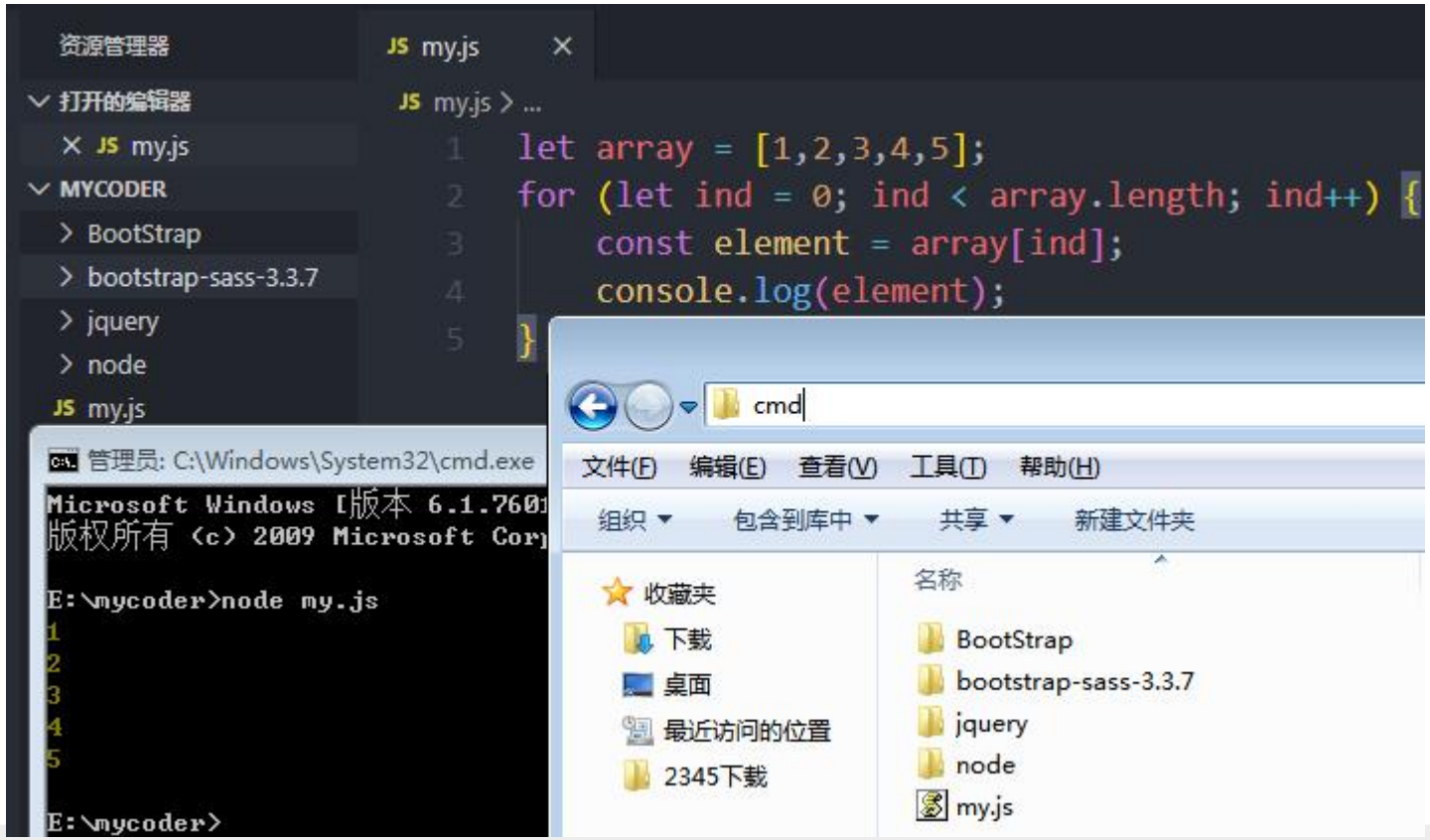
3. 输入cnpm –v检查是否正常： 记得重启DOS窗口

```
D:\nodejs>cnpm -v
cnpm@6.1.0 (D:\nodejs\node_global\node_modules\cnpm\lib\parse_a
npm@6.9.0 (D:\nodejs\node_global\node_modules\cnpm\node_modules
node@10.16.0 (D:\nodejs\node.exe)
npminstall@3.22.1 (D:\nodejs\node_global\node_modules\cnpm\node
ll\lib\index.js)
prefix=D:\nodejs\node_global
win32 x64 6.1.7601
registry=https://r.npm.taobao.org
```

执行js文件

执行你的第一个nodejs文件：

1. 创建一个nodejs文件，后缀为js，如：my.js，写入js代码;
2. 在当前路径打开DOS窗口；
3. node my.js 执行该文件；



第40章 Http服务器

华清远见-成都中心-H5教学部

目录

node.js内置模块

一、在nodejs的世界里，一个js文件就是一个模块；

二、默认情况下模块与模块之间是相互隔离的；

三、一个模块要使用别的模块，则必须要引用才能使用它；

四、模块分类

(1)nodejs内置模块： url、http、fs、querystring

(2)自定义模块

(3)第三方模块

五、使用内置模块

(1)内置模块是nodejs天生的

(2) 使用require引入后即可使用

示例:

```
var http = require("http");
var fs = require("fs");
var querystring = require('querystring')
```

导入node.js内置模块

六、官方文档：http://nodejs.cn/api/

自定义模块

模块之间可以相互引用，引入即执行，引入自定模块时，推荐使用绝对物理路径，这样程序在执行时就与命令所在目录无关了，不会发生模块找不到的错误。

自定义模块-exports暴露数据

1. 引用模块时，引用模块中的代码就会执行，但是如果模块中数据(变量或方法)怎样被使用呢?

2. 模块中数据只有被暴露后，被其它模块引入才能被使用。

3. 模块中数据被暴露的方式有两种：
   exports.自定名=数据名
   module.exports=数据名

自定义模块-exports暴露数据

nodejs_exports
- main.js
- one.js

main.js

```
var one = require("./one.js");
one.showName("王大");
console.log("one.js中total的值为:"+one.total);
one.showNum(100);
```

one.js

```
var total = 10;
function showName(name) {
    console.log("我的名字是"+name);
}
exports.showNum = function(num){
    var totalNum=0;
    for(var i=0;i<=num; i++) {
        totalNum += i;
    }
    console.log("0+1+....+"+num+"的值:"+totalNum)
}
exports.total = total;
exports.showName = showName;
```

```
PS E:\temp\nodejs_exports> node main.js
我的名字是王大
one.js中total的值为:10
0+1+....+100的值:5050
```

自定义模块-module.exports暴露数据

只适用于自定义模块中只有一个数据

Person.js

```
function Person(name,sex) {
    this.name=name;
    this.sex=sex;
    this.show=function() {
        console.log("我的name是"+this.name+",我的sex是"+this.sex);
    }
}
module.exports=Person;
```

main.js

```
var Person = require("./Person.js");
var person = new Person("王大","男");
person.show();
```

自定义模块-module.exports暴露数据

show.js

```
function show() {
    console.log("这是show.js中的");
}
module.exports=show;

/**
或
module.exports=function(){
    console.log("这是show.js中的");
}
**/
```

```
var show = require("./show.js");
show();
```

第一个Http服务器

http模块是node.js内置模块，使用它可以完成Http服务器的创建

```
//导入http模块
var http = require("http");
//创建一个http服务
var server = http.createServer(function(req,res){
    var xx = "<H1>Hello World</H1>";
    //响应客户端数据并关闭连接
    res.end(xx);
});
//指定指用当前电脑中的4000端口启动http服务，并处于监听中...
//第三个参数是函数，在成功启动服务器后回调
server.listen(4000,"localhost", function(){
    console.log("console.log("服务器开启，等待客户端访问...");");
});
```

```
D:\mynodejs\nodejsdemo>node 2.js
服务器开启，等待客户端访问...
```

第一个Http服务器

■node.js代码运行在服务器端；

■响应结束后，连接就会断开；

■启动服务器时使用server.listen(端口,ip, 回调函数)方法，该方法中ip可缺省，缺省值为当前电脑ip, 回调函数可缺省；

乱码处理

如果响应的数据有中文，则客户端会得到中文乱码，解决办法是使用res设置Mime类型

```
var http = require("http");
var server = http.createServer(function(req,res){
    //处理中文乱码
    res.setHeader("Content-Type","text/html;charset=UTF-8");
    var xx = "<H1>你好！世界的nodejs</H1>";
    res.end(xx);
});
server.listen(4000);
```

响应对象res的end和write方法

res.write(string)输出数据到客户端。

res.end(string)输出数据到客户端，并且结束输出。

第一次响应输出最后一定要结束。

```
var http = require("http");
var server = http.createServer(function(req, res) {
    res.setHeader("Context-Type", "text/html;charset=UTF-8");
    res.write("<!DOCTYPE html>");
    res.write("<html>");
    res.write("<head>");
    res.write("<meta charset='UTF-8'>");
    res.write("<title></title>");
    res.write("</head>");
    res.write("<body>");
    res.write("<h1 align='center'>这是一个write与end的区别</h1>");
    res.write("</body>");
    res.write("</html>");
    res.end("");//输出的最后一定要结束
});
server.listen(4000);
```

第41章 全局对象&文件&事件

华清远见-成都中心-H5教学部

目录

全局对象

Node.js中有一个特殊的对象，称为全局对象（Global Object），它及其所有属性都可以在程序的任何模块访问。

为什么node.js中会有全局变量---因为node.js模块与模块之间是相关隔离的

__finename:表示当前文件物理路径
__dirname:表示当前文件或文件夹所在目录

let t = setTimeout(cb, ms)
指定的毫秒(ms)数后执行指定函数(cb) 返回一个代表定时器的句柄值t。
clearTimeout(t) 取消定时器。

let t = setInterval(cb, ms)
指定的毫秒(ms)数后执行指定函数(cb),并且会定时重复执行。返回一个代表定时器的句柄值t。
clearInterval(t) 取消定时器

文件模块

导入文件模块使用 var fs = require(“fs”);

文件模块有很多函数，基本所有函数都有异步IO和同步IO两个版本。开发中常使用异步版本，使用异步版本时都有一个回调函数来监听异步IO操作的结束事件，并且回调函数通过参数获取异步IO操作的结果。

文件模块-读取文件内容

异步读取文件内容

fs.readFile(“文件路径”,回调函数callback)：

文件路径可以是任何文件类型的路径；

callback有两个参数,第一个参数用于获取异常,第二个参数用于接收读取的文件内容；

同步读取文件内容

fs.readFileSync(“文件路径”),

返回值为文件内容；

```
var fs = require("fs");

// 异步读取当前目录中的input.txt内容
fs.readFile(__dirname+'/input.txt', function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("异步读取: " + data.toString());
});

// 同步读取当前目录中的input.txt内容
var data = fs.readFileSync(__dirname+'/input.txt');
console.log("同步读取: " + data.toString());

console.log("程序执行完毕。");
```

事件循环

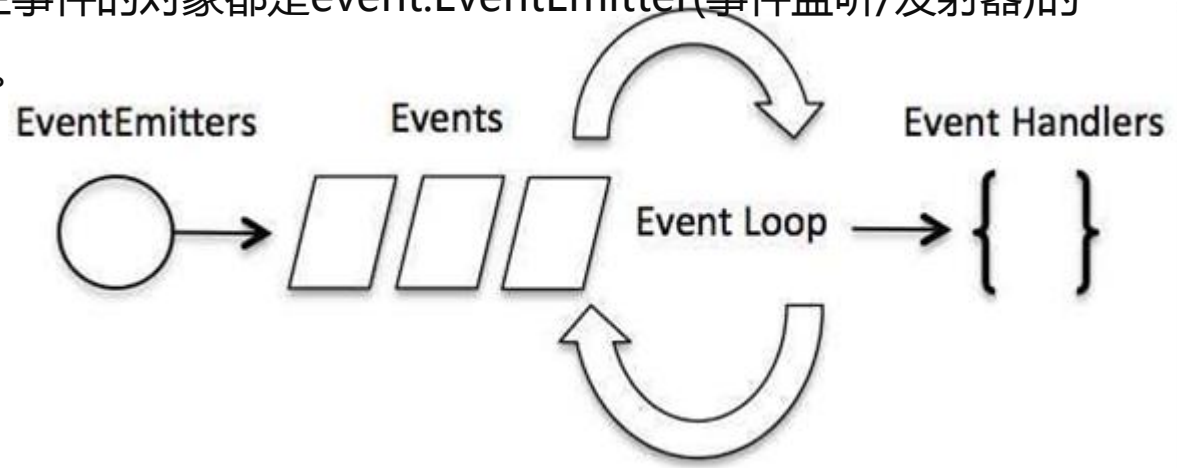
异步读取文件内容

NodeJS不是在各个线程为每个请求执行所有的工作，它是把工作添加到事件队列中，然后有一个单独的线程运行一个事件循环把这个工作提取出来。事件循环抓取事件队列中最上面的条目，执行它，然后抓取下一个条目。当执行到长期运行或有阻塞I/O的代码时，它不是直接调用该函数，而是把函数同一个要在此函数完成后执行的回调函数一起添加到事件队列中。当NodeJS事件队列中的所有事件都被执行完成时，nodejs应用程序终止。

Node.js中，所有异步的I/O操作，在完成的时候都会发送一个事件到事件队列中。

Node.js中的许多对象也都会分发事件，比如：net.Server 对象会在每次有新链接时分发一个事件；fs.readStream 对象会在文件被打开的时候分发一个事件等等，所有这些产生事件的对象都是event.EventEmitter(事件监听/发射器)的实例。我们可以通过“ require('events') ”来访问该模块..

事件回调的关键就是事件轮询。



事件循环

```
// 引入 events 模块
var events = require('events');
// 创建 eventEmitter 对象
var eventEmitter = new events.EventEmitter();
// 创建事件处理程序
var connectHandler = function connected() {
    console.log('事件处理函数监听到了');
}
// 绑定 www 事件处理程序，事件名可以任意
eventEmitter.on('www', connectHandler);
// 触发 www 事件
eventEmitter.emit('www');
```

第42章 路由

华清远见-成都中心-H5教学部

目录

url解析

路由: 客户端向服务器发出请求时,会有很多不同的url, 服务器根据不同的url作出不同的响应

解析请求的url时,要使用url模块

```
let url = require("url");
let urlObj = url.parse(req.url);//req是请求对象
let urlPath = urlObj.pathname;
```

得到的urlPath不会包含请求参数

示例：

http://localhost:9999	urlPath的值为 /
http://localhost:9999/user.do	urlPath的值为 user.do
http://localhost:9999/user.do?name=小明&pwd=123	urlPath的值为 user.do

路由

```
var http = require("http");
var url = require("url");
var server = http.createServer(function(req, res) {
    //处理中文乱码
    res.setHeader("Content-Type", "text/html;charset=UTF-8");
    var urlObj = url.parse(req.url);
    var urlPath = urlObj.pathname;
    if (urlPath == "/") {
        res.end("这是中国");
    } else if (urlPath == "/bj") {
        res.end("这是北京");
    } else if (urlPath == "/sh") {
        res.end("这是上海");
    } else {
        res.end("没有找到资源");
    }
});
server.listen(9999);
```

路由

```
var http = require("http");
var fs = require("fs");
var url = require("url");
var server = http.createServer(function(req, res) {
    res.setHeader("Content-Type", "text/html;charset=UTF-8");
    var urlObj = url.parse(req.url);
    var urlPath = urlObj.pathname;
    if(urlPath  == "/") {
        fs.readFile("./page/cn.html",function(err, fileContent){
            if(err) {
                res.end("读取文件异常");
            }
            res.end(fileContent);
        });
    } else if(urlPath  == "/bj") {
        fs.readFile("./page/bj.html",function(err,fileContent){
            if(err) {
                res.end("读取文件异常");
            }
            res.end(fileContent);
        });
    } else {
         res.end("没有找到资源");
    }
});

server.listen(9999);
```

客户端常见请求方式

1.在浏览器中输入网址发出get请求；

2.css,js,img自动发出get请求；

3.超连接发出get请求

得到用户(get)；

4.js发出get请求

window.location.href= ‘url?name=wgr&pwd=123’ ；

5.表单提交发出请求(get或post)；

6.ajax请求(get或post)—后期内容会讲解；

客户端常见请求方式

表单请求示例

```
<form action="/login.do" method="get">
    用户名:<input type="text" name="name"/><br  />
    密码:  <input type="password" name="pwd"/><br  />
    <input type="submit" value="提交"/>
</form>

<form action="/login.do" method="post">
    用户名:<input type="text" name="name"/><br  />
    密码:  <input type="password" name="pwd"/><br  />
    <input type="submit" value="提交"/>
</form>
```

Get与Post的区别

1. 语义及使用场景：
   ① GET：获得，得到，所以更多的用于获取信息，如：user?uid=102 获取用户信息，news?nid=10获取新闻；
   ② POST：邮递，邮寄，把...放入，所以更多的用于提交信息，如添加用户信息、添加新闻信息等；

2. 数据提交方式：
   ① GET请求我们熟知的数据提交方式是url?a=1&b=2&c=3的方式来提交数据的,键值之间使用=连接，多个值之间使用&符号连接；对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；针对上面的url地址，服务器收到了三个信息：
   键名分别是a、b、c,值分别为1、2、3；

   ② POST请求我们熟知的数据提交方式是：浏览器先发送header，服务器响应100（continue），然后再发送data，服务器响应200（返回数据）；

Get与Post的区别

3. 数据提交大小：

根据HTTP协议规范，并没有对请求头和请求体提出长度要求 ；我们熟知的GET请求有大小限制的原因大概出自以下三点：

① 浏览器：浏览器对URL地址长度会有限制，所以，自然的，通过URL传值的GET请求自然也就有了大小限制；

② 服务器：一般来讲，处于安全、处理能力、性能等各方面考虑，服务器会对请求的url地址长度有限制，不过这个与是GET还是POST没有关系，也就是说POST请求并不是随便多大都行，本身也是有大小限制的；

③ 内容传播：GET更多用于获取数据，如果传递的内容过大，url地址就会很长，不方便url的传播；

Get与Post的区别

4. 数据提交的安全性：

本质上安全性是一样的：

① GET是通过URL方式发起请求并提交数据的，可直接看到，明文传输；

② POST是通过请求体body提交数据，开发者工具或者抓包工具可以看到，同样是明文传输；

③ GET请求默认会保存在浏览器历史纪录中，会被别人看到；

Get与Post的区别

5. 对服务器状态的影响：

在主观意识上：

① GET是获取数据，一般不会修改服务器上的数据；

② POST是提交数据，主观意识上认为修改了服务器上的数据；

实际情况有没有修改，则完全由服务器绝对，比如get一个新闻信息时修改他的访问次数；

处理请求-判断请求方法

let method = req.method.toUpperCase(); //req是请求对象；

当服务器接收到的是get请求时， method的值为”GET” ；

当服务器接收到的是post请求时，method的值为”POST” ；

服务器判断请求方式有何用---对于不同的请求方式服务器在解析请求参数时必须采取不一样的方式；

处理请求-获取get请求数据

请求参数都是“键值对”，获取时就根据键取值即可

```
let url = require("url");
let parseObj = url.parse(request.url, true);
let query = parseObj.query;
//decodeURI是处理中文转码
let name = decodeURI(query.name);          //name是参数的键名
let pwd = decodeURI(query.pwd);            //pwd是参数的键名
```

处理请求-获取post请求数据

1. 发请post请求时带参数：

```
<form action="/login.do" method="post">
    用户名:<input type="text" name="name"/><br  />
    密码:  <input type="password" name="pwd"/><br  />
    <input type="submit" value="提交"/>
</form>
```

2. 获取post请求参数进，要使用内置模块”querystring” ：

```
var querystring = require('querystring');
.....
var body = "";
req.on('data', function(chunk) {
    body += chunk;
});
req.on('end', function() {
    body = querystring.parse(body); //将一个字符串反序列化为一个对象
    var name = body.name; //name是参数的键名
    var pwd = body.pwd; //pwd是参数的键名
    res.end("成功");//注意响应代码的位置，  因为这是异步操作
});
```

第43章 NPM

华清远见-成都中心-H5教学部

目录

npm是什么

npm：NodePackageManager，nodejs包管理器

npm管理模块

初始化：

npm init 初始化nodejs项目，会生成package.json文件；

安装模块：

1. npm install <packagename> 安装模块到当前文件夹下的node_modules文件夹下；
2. npm i <packagename> 同上，简写；
3. npm i <packagename1 packagename2 > 同上，只是一次安装多个第三方模块；
4. npm i packagename –-save 同上，这样安装的模块会作为依赖包追加到package.json文件里面；
5. npm i packagename –g 这样安装的模块是全局模块，在任何地方都可以使用；
6. npm i packagename@version 默认安装最新版本，这样可以安装指定版本；

卸载模块：

npm uninstall packagename 卸载指定模块

更新模块:

npm update packagename 不指定版本号直接重新安装一次，也是更新

海量视频 贴身学习

超多干货 实时更新

THANKS

谢谢